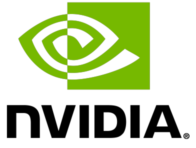


NVIDIA DOCA DPI

Programming Guide

MLNX-15-060466 _v1.4 | September 2022

Table of Contents

Chapter 1. Introduction........ 1
Chapter 2. Prerequisites........ 2
Chapter 3. Architecture........ 3
3.1. Signature Database........ 3
3.2. DPI Queue........ 3
Chapter 4. API........ 5
4.1. doca_dpi_config........ 5
4.2. doca_dpi_load_signatures........ 5
Chapter 5. Flow Life Cycle........ 6
Chapter 6. DOCA DPI gRPC........ 7
6.1. Proto-buff........ 8
6.2. Usage........ 9
6.3. gRPC API........ 9
6.3.1. doca_dpi_ctx........ 10
6.3.2. DocaDpiFlowCtx........ 10
6.3.3. DocaDpiInitParams........ 10
6.3.4. DocaDpiInitResponse........ 10
6.3.5. DocaDpiFlowCreateParams........ 11
6.3.6. DocaDpiGenericPacket........ 11
6.3.7. DocaDpiDequeueParams........ 11
6.3.8. DocaDpiErrorInfo........ 11
6.3.9. DocaDpiActionResponse........ 12
6.3.10. DocaDpiLoadSignaturesParams........ 12
6.3.11. DocaDpiInit........ 12
6.3.12. DocaDpiLoadSignatures........ 13
6.3.13. DocaDpiDestroy........ 13
6.3.14. Additional gRPC APIs........ 13
6.4. Multi-Processing and Multithreading........ 13
6.4.1. Enqueue........ 13
6.4.2. Matching........ 14
6.5. DOCA DPI gRPC Client API........ 14
6.5.1. doca_dpi_grpc_generic_packet........ 14
6.5.2. doca_dpi_config_t........ 15
6.5.3. General Function Signatures........ 15

6.5.4. doca_dpi_grpc_enqueue......15
6.5.5. doca_dpi_grpc_flow_destory and doca_dpi_grpc_flow_match_get......16
6.5.6. Pre-use Setup......16
6.5.7. Destroying gRPC Client......16

Chapter 1. Introduction

Deep packet inspection (DPI) is a method of examining the full content of data packets as they traverse a monitored network checkpoint.

DPI provides a more robust mechanism for enforcing network packet filtering as it can be used to identify and block a range of complex threats hiding in network data streams more accurately. This includes:

- Malicious applications
- Malware data exfiltration attempts
- Content policy violations
- Application recognition
- Load balancing

The DOCA DPI library supports inspection of the following protocols:

- HTTP 2.0/1.1/1.0
- TLS/SSL client hello and certificate messages
- DNS
- FTP

TCP/UDP stream-based signatures may detect applications on other protocols.

Chapter 2. Prerequisites

DPI-based applications can run either on the host machine, or on the NVIDIA® BlueField® DPU target. Since the DPI leverages the Regular Expressions (RegEx) Engine, users must make sure it is enabled.

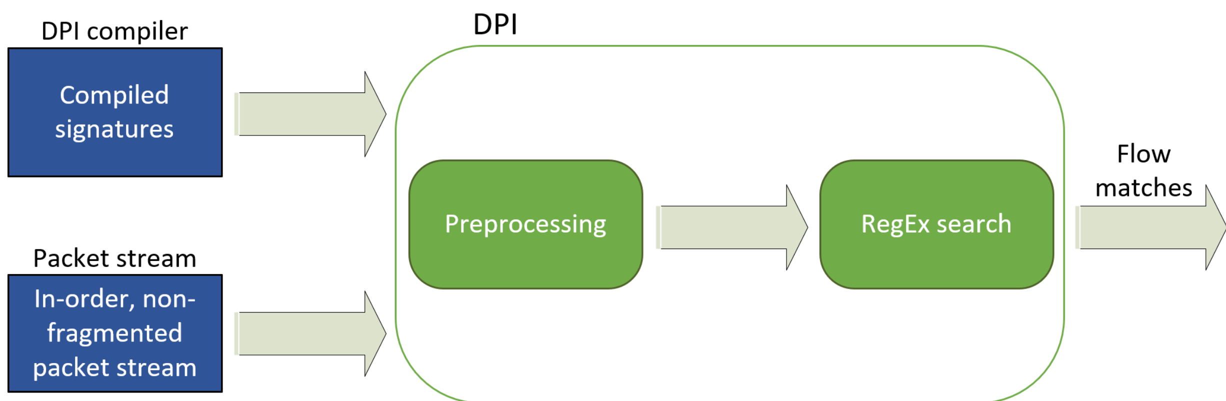
The RegEx engine is enabled by default on the DPU. To use DPI directly on the host, run:

```
host> sudo /etc/init.d/openibd stop
dpu> echo 1 > /sys/class/net/p0/smart_nic/pf/regex_en
dpu> cat /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
400
# Make sure to allocate 200 additional hugepages
dpu> sudo echo 600 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
dpu> systemctl start mlx-regex
# Verify the service is properly running
dpu> systemctl status mlx-regex
host> sudo /etc/init.d/openibd start
```

Note: Commands with the `host` prompt must be run on the host. Commands with the `dpu` prompt must be run on BlueField (Arm).

Chapter 3. Architecture

The following diagram shows how the DPI library receives for processing a compiled signatures files and a stream of "in-order" and non-fragmented packets, and returns matches per flow.



3.1. Signature Database

The signature database is compiled into a CDO file by the DPI compiler. The CDO file includes:

- Compiled RegEx engine rules
- Other signature information

The application may load a new database while the DPI is processing packets.

For more information on the DPI compiler, please refer to the NVIDIA DOCA DPI Compiler document.

3.2. DPI Queue

A DPI queue is designed to be used by a worker thread. The DPI queue holds the flow's state. Therefore, all packets from both directions of the flow must be submitted to the same DPI queue in-order. Each packet must be injected along with a flow context and a direction.

A flow direction is usually represented by a 5-tuple, but it can also be a 3-tuple for other protocols.

Note: From the application's side, the connection tracking (CT) logic must handle out-of-order packets as well as fragmented packets. Once a connection has timed out or terminated, the application must notify the DPI library as well.

Chapter 4. API

For the library API reference, refer to DPI API documentation in NVIDIA DOCA Libraries API Reference Manual.

Note: The pkg-config (`*.pc` file) for the DPI library is named `doca-dpi`.

The following sections provide additional details about the library API.

4.1. doca_dpi_config

The following is the DPI initialize configuration struct. The library is configured according to the struct fields.

```
struct doca_dpi_config_t {
            uint16_t nb_queues;
            uint32_t max_packets_per_queue;
            uint32_t max_sig_match_len;
};
```

nb_queues
The number of DPI queues to be used.

max_packets_per_queue
The number of packets concurrently processed by the DPI engine.

max_sig_match_len
The maximum length that DPI guarantees to provide a match on, including across consecutive packets.

4.2. doca_dpi_load_signatures

Before enqueueing packets for processing, the DPI library must be loaded with signatures by the main thread:

```
int doca_dpi_load_signatures(doca_dpi_ctx* ctx, const char* cdo_file);
```

ctx [in]
Pointer to DPI opaque context struct created by `doca_dpi_init()`.

cdo_file [in]
`*.cdo` file path. The file is created by the DPI compiler according to the Suricata rules that have been provided to it.

Returns
0 on success, and an error code otherwise.

Chapter 5. Flow Life Cycle

1. Once a new flow is detected by the connection tracking software, the user should call `doca_dpi_flow_create()`.
2. Every incoming packet classified for this flow should be enqueued by calling `doca_dpi_enqueue()`.

Note: For every mbuf injected, the user is not allowed to free the mbuf until the mbuf is dequeued.

Note: If an external attach is used, users must follow the DPDK guidelines for `rte_pktmbuf_attach_extbuf()` to make sure the mbuf is freed when both the user and the DPI free the mbuf.

3. To poll for the results, the application must call `doca_dpi_dequeue()`. The result will contain matching information (if matched).
4. When the connection tracking software detects that the flow is terminated or aged-out, the application should notify the DPI library by calling `doca_dpi_flow_destroy()`.

Chapter 6. DOCA DPI gRPC

This section describes the gRPC (Google remote procedure calls) support for DOCA DPI API. The DOCA DPI gRPC-based API, allows users on the host to leverage the HW offload capabilities of the BlueField-2 DPU using gRPC calls from the host itself. For more information about gRPC support in DOCA, refer to the NVIDIA DOCA gRPC Infrastructure User Guide.

The following figure illustrates the DOCA DPI gRPC server-client communication.

Host (x86)
DOCA DPI gRPC Client
DOCA gRPC Orchestrator Client
Traffic from network
vf0
pf0
Physical Function (PF)
BlueField
pf0hpf
pf0vf0
OVS-gRPC
DOCA gRPC Orchestrator Daemon
SF0
SF1
SF1-Rep
DOCA DPI gRPC Server
Arm

Note: The gRPC DPI server needs an SF dedicated only to holding mbuf packets. This SF must be different from the SF used for the control path between host and server.

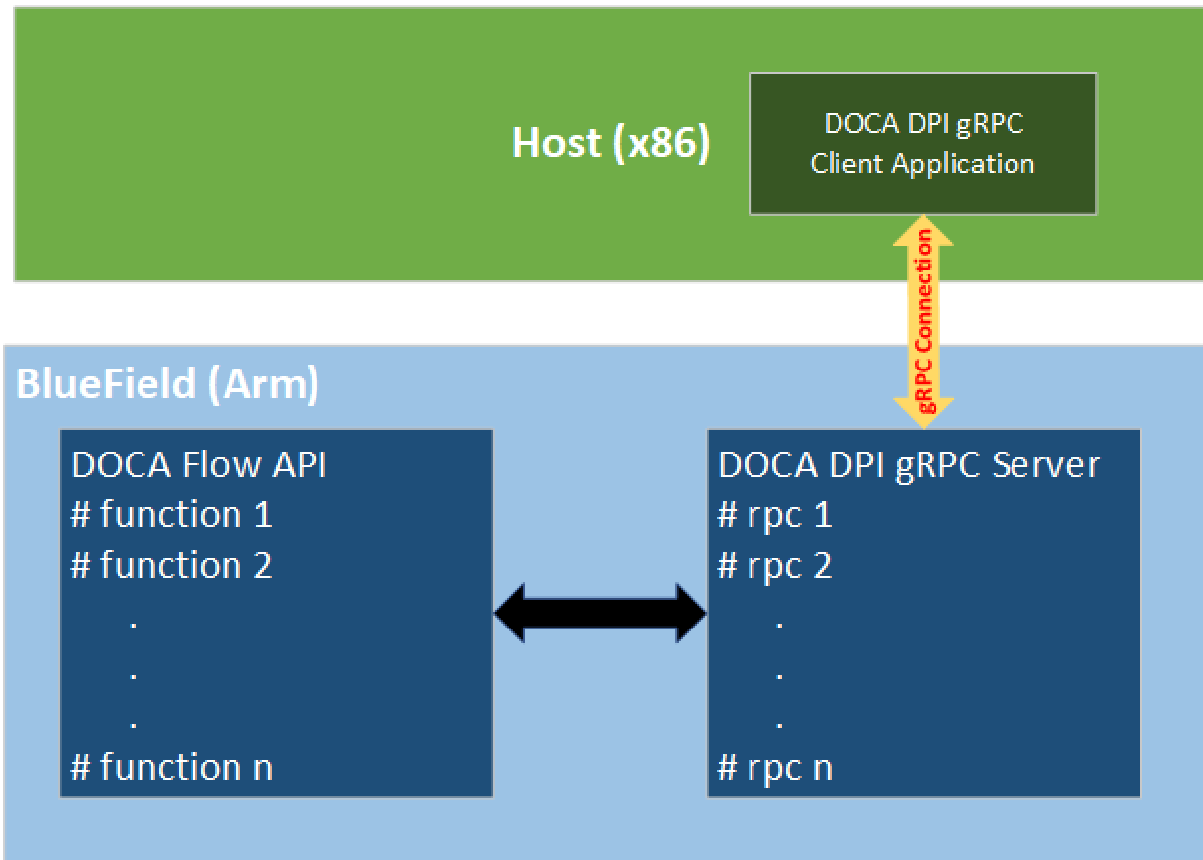
6.1. Proto-buff

As with every gRPC proto-buff, DOCA DPI gRPC proto-buff defines the service it introduces and the messages used for communication between the client and the server.

Users are provided with the proto-buff file `doca_dpi.proto`. This file defines message representation of DOCA DPI API structs and enums, and defines the service, its RPCs, the request, and response objects.

Each message defined in `doca_dpi.proto` with the `DocaDpi` prefix represents exactly one struct or enum defined by the DOCA DPI API.

The following figure illustrates how DOCA DPI gRPC server represents the DOCA DPI API.



The proto-buff path for DOCA DPI gRPC is `/opt/mellanox/doca/infrastructure/doca_grpc/doca_dpi/doca_dpi.proto`.

6.2. Usage

Similarly to the regular DPI API, the gRPC DPI is dependent on the same constraints, and its RPCs must be called in the same manner, except for `doca_dpi_init` which is invoked by the service upon loading instead of via gRPC call.

The host must use a connection tracking mechanism to provide the DOCA DPI gRPC service with ordered and unfragmented packets.

The gRPC API replaces the usage of `struct rte_mbuf` packets with `message DocaDpiGenericPacket` which is not protocol-dependent and is detailed in section Enqueue.

For more information about the sections that follow, please refer to the regular DPI API guide (chapters 1-5) or the `doca_dpi.h` file.

6.3. gRPC API

Refer to NVIDIA DOCA Libraries API Reference Manual for DPI API documentation.

The following subsections provide additional details about the library gRPC API.

6.3.1. doca_dpi_ctx

Unlike direct usage of DOCA DPI API, no `struct doca_dpi_ctx` is passed to the client. Instead, it is saved in the service and used when needed.

The client API does provide `struct doca_dpi_ctx*`, but it is only a symbolic identifier of the actual struct, `doca_dpi_ctx`, on the server. This means that the client cannot configure the DPI engine with doca_dpi_config. Instead, when the server is deployed it uses a default configuration.

6.3.2. DocaDpiFlowCtx

`DocaDpiFlowCtx` contains a unique ID to identify the real flow context instance:

```
message DocaDpiFlowCtx {
 uint64 unique_id = 1;
}
```

unique_id
: Unique identifier given by the server to identify an existing flow.

6.3.3. DocaDpiInitParams

`DocaDpiInitParams` is the gRPC equivalent to doca_dpi_config_t. It contains a unique ID to identify the real flow context instance:

```
message DocaDpiFlowCtx {
    uint32 uint16_nb_queues = 1;
    uint32 max_packets_per_queue = 2;
    uint32 max_sig_match_len = 3;
}
```

uint16_nb_queues
: The number of DPI queues to create.

max_packets_per_queue
: The maximum number of packets a single queue can have concurrently processed by the DPI engine.

max_sig_match_len
: The maximum length that DPI guarantees to provide a match on, including across consecutive packets.

6.3.4. DocaDpiInitResponse

```
message DocaDpiInitResponse {
    DocaDpiErrorInfo erreno = 1;
    uint32 nb_microservices = 2;
}
```

erreno
: Possible error info.

nb_microservices
: Number of active micro services with independent IP address. The i-th service listens to the given IP and the given port+i.

6.3.5. DocaDpiFlowCreateParams

This structure is used to replace the input arguments of `doca_dpi_flow_create`.

The parameter `uint16_dpi_q` can be anything between 0 and the number of `lcores` the DPU has minus 1.

```
message DocaDpiFlowCreateParams {
    DocaDpiParsingInfo parsing_info = 1;
    uint32 uint16_dpi_q = 2; /* The DPI packets queue the flow will be assigned to.
 */
}
```

parsing_info
: Holds the connection information (e.g. source IP, source port, destination IP, destination port, flow direction).

uint16_dpi_q
: Defines with which packet processing queue to associate the entire flow and its packets.

6.3.6. DocaDpiGenericPacket

```
message DocaDpiGenericPacket {
 bytes segment = 1;  /* The packet data, max length is 65535 (0xffff). */
}
```

segment
: Sequence of bytes of the original received packet. Including the packet header in bytes is not mandatory.

The proto-buff type `DocaDpiGenericPacket` is used to enqueue a packet with up to 65535 bytes of payload or segment (both are applicable) which can include the actual packet headers.

6.3.7. DocaDpiDequeueParams

This structure is used to replace the input arguments of `doca_dpi_dequeue`.

When dequeuing a packet, the mandatory parameter `uint16_dpi_q` refers to the inner packets-queue from which to dequeue a packet. `uint16_dpi_q` can be anything between 0 and the number of lcores the DPU has minus 1.

```
message DocaDpiDequeueParams {
 uint32 uint16_dpi_q = 1; /* The DPI queue from which to dequeue the flows packets.
 */
}
```

uint16_dpi_q
: The packet processing queue from which to dequeue a previously inserted packet.

6.3.8. DocaDpiErrorInfo

This structure is used to indicate any DOCA DPI return status from the actual method found in `doca_dpi.h`. Also, if the remote DOCA DPI API prompts a message to the terminal, even if no error occurs, the string field `err_msg` will contain that message.

Errors are returned as follows:

```
message DocaDpiErrorInfo {
```

```
 int64 error_code = 1;
 optional string err_msg = 2;
}
```

error_code
The return status from the corresponding non-remote API. The gRPC header has documentation of possible values for each API call. An `error_code` with non-zero value indicates that an error occurred.

err_msg
(Optional) Exists only if a print had occurred on the remote server.

An error code with a negative value indicates that an error occurred. A positive value returned indicates a DPI enum value (enum type depends on the API call).

6.3.9. DocaDpiActionResponse

This structure is used to replace the return value and set arguments by the remote `doca_dpi_dequeue` and `doca_dpi_flow_match_get`.

```
message DocaDpiActionResponse {
 DocaDpiErrorInfo erreno = 1;
 DocaDpiResult result = 2;  /* gRPC message equvalent to doca_dpi_result struct. */
}
```

erreno
Possible error info.

result
The DPI engine results from processing the packet/flow.

6.3.10. DocaDpiLoadSignaturesParams

```
message DocaDpiLoadSignaturesParams {
 oneof cdo {
  string cdo_filename = 1;  /* Path, on the DPU, to a cdo file */
  bytes cdo_data = 2;       /* Content of a cdo file. */
 }
}
```

cdo_filename
An absolute path of the DPU to a `.cdo` file.

cdo_data
The content of a `.cdo` file, as a sequence of bytes.

Only one of the fields can be present, similarly to a Union in C.

The CDO file must be produced in the same manner as in regular DPI API.

6.3.11. DocaDpiInit

```
rpc DocaDpiInit (DocaDpiInitParams) returns (DocaDpiInitResponse);
```

Creates a new DPI context on the server. It aborts with gRPC status code ALREADY_EXISTS if a context already exists. For instructions about destroying it and creating a new one, see DocaDpiDestroy.

Note: This RPC must be called before any other RPC.

6.3.12. DocaDpiLoadSignatures

```
rpc DocaDpiLoadSignatures (DocaDpiLoadSignaturesParams) returns (DocaDpiErrorInfo);
```

This differs from the regular `doca_dpi_load_signatures` by including the option to send a CDO file, instead of only a path to a CDO file, on BlueField:

- Like the regular `doca_dpi_load_signatures` argument, the argument `DocaDpiLoadSignaturesParams.cdo_filename()` is a path to a CDO file on the DPU
- Unlike the regular `DocaDpiLoadSignaturesParams.doca_dpi_load_signatures()`, an entire CDO file can be sent inside `cdo_data` instead of using a path

6.3.13. DocaDpiDestroy

The service can properly exit by calling the RPC method `DocaDpiDestroy`.

```
rpc DocaDpiDestroy (DocaDpiDestroyParams) returns (DocaDpiDpiDestroyResponse);
```

When invoked, the server destroys the DPI context and releases all state resources. It does not destroy any of the existing DPI flows and does not free enqueued packets.

Note: Make sure to dequeue all packets and destroy all DPI flows before calling this method.

6.3.14. Additional gRPC APIs

```
rpc DocaDpiDequeue (DocaDpiDequeueParams) returns (DocaDpiActionResponse);
rpc DocaDpiFlowCreate (DocaDpiFlowCreateParams) returns (DocaDpiFlowCreateResponse);
rpc DocaDpiFlowDestroy (DocaDpiFlowDestroyParams) returns
 (DocaDpiFlowDestroyResponse);
rpc DocaDpiFlowMatchGet (DocaDpiFlowMatchGetParams) returns (DocaDpiActionResponse);
rpc DocaDpiSignatureGet (DocaDpiSignatureGetParams) returns
 (DocaDpiSignatureGetResponse);
rpc DocaDpiSignaturesGet (DocaDpiSignaturesGetParams) returns
 (DocaDpiSignaturesGetResponse);
rpc DocaDpiStatGet (DocaDpiStatGetParams) returns (DocaDpiStatInfo);
```

These methods work in the same manner as the regular DPI API, where the input variables to the matching `doca_dpi.h` method are members of the `<Prefix>Params` structure, and the output variables (i.e., return value and pointers to be set) are returned in the response.

6.4. Multi-Processing and Multithreading

6.4.1. Enqueue

```
rpc DocaDpiEnqueue (DocaDpiEnqueueParams) returns (DocaDpiErrorInfo);
```

Enqueue RPC is thread-safe per queue. This means that:

- Multiple processes/threads can use enqueue when the flows are linked to different queues
- Multiple processes/threads cannot use enqueue when the flows are linked to the same queue

6.4.2. Matching

```
rpc DocaDpiDequeue (DocaDpiDequeueParams) returns (DocaDpiActionResponse);
rpc DocaDpiFlowMatchGet (DocaDpiFlowMatchGetParams) returns (DocaDpiActionResponse);
```

The same DPI gRPC server can be used by many processes/threads by using one of the following 2 options:

- Use `DocaDpiFlowMatchGet` instead of `DocaDpiDequeue`. Notice that dequeuing is still needed to free up the DPI queue.
- Use a different `uint16_dpi_q` value for each process/thread. By doing so, users can make sure packets from only one processing entity reach a certain queue and that only its packets are dequeued from that queue.

There can be only one DPI server at a time due to the ownership of the HW RegEx accelerator.

6.5. DOCA DPI gRPC Client API

This section describes the recommended way for C developers to use gRPC support for DOCA DPI API.

For the library API reference, refer to DOCA DPI gRPC API documentation in NVIDIA DOCA Libraries API Reference Manual.

The following sections provide additional details about the library API.

The DOCA installation includes `libdoca_dpi_grpc` which is a library that provides a C API wrapper to the C++ gRPC while mimicking the regular DOCA DPI API for ease of use and to allow smooth transition to the Arm.

This library API is exposed in `doca_dpi_grpc_client.h` and is essentially the same as `doca_dpi.h` with the notation differences detailed in the following subsections.

Note: The gRPC client lib does not depend on DPDK.

Messages from the server are prompted to the terminal only if errors occur (i.e., only when the gRPC fails or remote DPI invocation returns an error).

6.5.1. doca_dpi_grpc_generic_packet

Packets are not of type `rte_mbuf` (DPDK). Instead, the following struct is defined:

```
struct doca_dpi_grpc_generic_packet {
    uint8_t *segment;  /**< The buffer with data to be scanned by the DPI */
    uint16_t seg_len;  /**< The length of the data inside segment buffer */
};
```

`segment`

Sequence of bytes that describe a packet or a packet payload (both are applicable).

`seg_len`

The number of bytes in the segment. A segment can contain an entire packet or just the payload (both are applicable).

The struct `struct doca_dpi_result` is replaced by `struct doca_dpi_grpc_result` which uses `doca_dpi_grpc_generic_packet` instead of `rte_mbuf`.

6.5.2. doca_dpi_config_t

Same as the regular DPI config structure with an additional field:

```
struct doca_dpi_config_t {
    uint16_t nb_queues;
    uint32_t max_packets_per_queue;
    uint32_t max_sig_match_len;
    const char *server_address;
};
```

nb_queues
Number of DPI queues.

max_packets_per_queue
The maximum number of packets a single queue can have concurrently processed by the DPI engine.

max_sig_match_len
The maximum length that DPI guarantees to provide a match on, including across consecutive packets.

server_address
String representing the server IP (e.g., "127.0.0.1" or "192.168.100.3:5050"). If no port is provided, it uses the server's default port.

6.5.3. General Function Signatures

All signature prefixes have changed so `doca_dpi_<name>` becomes `doca_dpi_grpc_<name>`. For example, the function `doca_dpi_init` is replaced with:

```
struct doca_dpi_ctx* doca_dpi_grpc_init(const struct doca_dpi_config_t *config, int
 *error);
```

config [in]
Configuration for the remote server.

error [out]
Output error. Negative value indicates an error.

The function returns a mock pointer to the remote DPI opaque context struct.

6.5.4. doca_dpi_grpc_enqueue

The function `doca_dpi_grpc_enqueue` has two additional parameters that `doca_dpi_enqueue` does not have:

```
int doca_dpi_grpc_enqueue(struct doca_dpi_flow_ctx *flow_ctx,
                 struct doca_dpi_grpc_generic_packet *pkt, bool initiator,
                 uint32_t payload_offset, void *user_data,
                 size_t user_data_len, uint16_t dpi_q);
```

flow_ctx [in]
The flow context handler.

pkt [in]
The packet as binary buffer to be processed.

initiator [in]
Indicates to which direction the packet belongs.
1 – if the packet arrives from client to server.

0 – if the packet arrives from server to client.

Note: Typically, the first packet arrives from the initiator (client).

`payload_offset [in]`
Indicates where the packet's payload begins.

`user_data [in]`
Private user data to be returned when the DPI job is dequeued

`user_data_len [in]`
Length of the `user_data` param.

`dpi_q [in]`
The DPI queue the flow was created on.

The function returns `doca_dpi_enqueue_status_t` or other negative error code.

Enqueue does not take ownership of packet memory (unlike regular DOCA DPI). This means that enqueued packets can be freed after call return, but they must be dequeued to be freed on the server.

6.5.5. doca_dpi_grpc_flow_destory and doca_dpi_grpc_flow_match_get

These functions are the same as their non-remote equivalents, `doca_dpi_flow_destory` and `doca_dpi_flow_match_get`, except for one additional parameter:

```
uint16_t dpi_q
```

`dpi_q [in]`
The DPI queue on which to create the flows.

6.5.6. Pre-use Setup

The client gRPC API should be used only after the gRPC server is spawned and finished initializing. For info about deployment, refer to NVIDIA DOCA gRPC Infrastructure User Guide.

Except for the minor differences mentioned in this chapter, the client gRPC API should be used in the same manner as the regular API as specified in previous chapters (i.e., using the same initialization flow, API calls and structs, limitations, etc).

The client gRPC API must not be used together with the regular DOCA DPI file (described in chapters 1-5), `doca_dpi.h` .

6.5.7. Destroying gRPC Client

Destroying the gRPC client does not destroy the gRPC DPI server. For information on how to destroy the server, refer to NVIDIA DOCA gRPC Infrastructure User Guide.

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation nor any of its direct or indirect subsidiaries and affiliates (collectively: "NVIDIA") make no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assume no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA, the NVIDIA logo, and Mellanox are trademarks and/or registered trademarks of Mellanox Technologies Ltd. and/or NVIDIA Corporation in the U.S. and in other countries. The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a world¬wide basis. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2022 NVIDIA Corporation & affiliates. All rights reserved.

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com

NVIDIA.